# LOBS: Load Balancing for Similarity Peer-to-Peer Structures

David Novák and Pavel Zezula

Masaryk University, Brno, Czech Republic
xnovak8@fi.muni.cz zezula@fi.muni.cz

**Abstract.** The concept of peer-to-peer structures has recently been applied on the problem of large-scale similarity search. This resulted in systems where the computational load of the peers is of a high importance. Since no current load-balancing technique is designed for structures of this kind, we propose LOBS – a general system for load-balancing in peer-to-peer structures with time-consuming searching. LOBS is based on the following principles: measuring the computational load, separation of the logical and the physical level of the system, and detailed analysis of the load source to exploit either data relocation or replication.
This work contains results of experiments we conducted using a prototype implementation of LOBS. In these trials, we used a real-life dataset and we varied the number of peers and the distribution of the query traffic in the system. The results show that LOBS is able to cope with any query-distribution and that it improves both the utilization of resources and the performance of the query processing. The costs of balancing are reasonable and are very small if there is time to adapt to a query-traffic. The behaviour of LOBS is independent of the network size.

## 1 Similarity Peer-to-Peer Structures

The real-life experience with the similarity search shows that this task is both difficult and very expensive in terms of processing time [19]. The peer-to-peer (P2P) structures seem to be a suitable solution for content-based retrieval in huge data collections [6]. They have potential to provide both an extensive data storage and a strong distributed searching engine.

In this work, we focus on P2P structures for a full-value content-based search [5, 9, 14], which typically requires a time-consuming local search at the peers involved in the query processing. Experiments show that distribution of the *computational load* generated by the similarity queries is skewed – even in systems with a fair data distribution and a uniform query traffic. Moreover, the real-life traffic tends to have Zipfian distribution, which makes the situation even worse – see Figure 1 for an example. This waste of resources and degradation of search performance call for improvement.

The issue of keeping the load distribution fair has been considered in all types of P2P structures. The load may be defined in various ways. Majority of balancing techniques [2, 10, 8, 3, 12, 11] focus on balancing of the data volume

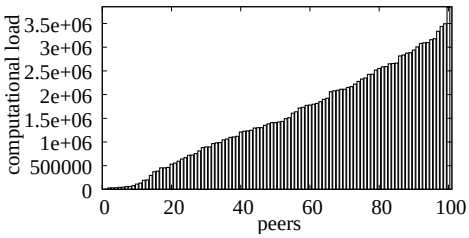


**Fig. 1.** Example of a computational-load distribution without any load balancing.

stored by individual peers. Some balancing mechanisms [18] measure (or may theoretically measure [10, 11]) the number of query accesses. In the similarity P2P structures, the actual time of the query processing is varying and depends on the particular query, on size of the local data, on quality of the local index, and on other fluctuating and hardly predictable variables. Therefore, it is necessary to consider the very computational load of the peers.

The similarity P2P structures based on general data models [19], e.g. $GHT^*$ [5] or $VPT^*$ [6], do not have a linear network topology nor manage a sortable data domain. This means that data cannot be simply shifted from a node to its neighbour which is a very natural action often used by current balancing techniques. All these facts made us design LOBS – a new balancing technique for similarity peer-to-peer networks.

**Problem Formulation, Our Approach, and Its Main Characteristics** The goal of this work is to propose a load-balancing mechanism which should:

- consider the computational load caused by the search requests evaluation,
- take into account structures with non-linear topology (e.g. trees),
- be fully decentralized.

The presented system LOBS is based on the following principles:

- it separates the logical and the physical level of the system,
- it exploits data relocation and data replication,
- it analyzes the source of the load to apply a suitable balancing operation.

The results show that:

- LOBS is able to cope with any query-distribution,
- it improves both the resources utilization and the system search performance,
- the costs of balancing are reasonable compared to the level of imbalance and are very small if the system has time to adapt to a query-traffic,
- the behaviour of LOBS is independent of the size of the network.

**Paper Structure** Section 2 maps current achievements in the area of load-balancing in structured P2P networks. In Section 3, we introduce the balancing system LOBS. Section 4 analyzes LOBS on the basis of experiments with a prototype implementation of LOBS. The paper concludes in Section 5. Detailed algorithms and some experiments' results omitted due to the space limitations can be found in a technical report [15].

## 2 Related Work

Majority of recent works in the area of P2P load balancing [2, 10, 8, 3, 12, 11] is motivated by the need of efficient processing of interval queries. The load-measure considered by these techniques is either the *amount of data* stored by individual peers or the *number of accesses* per peer since the data I/O-costs are the most important aspect to be considered for the interval queries. Some of these techniques [2, 8] are designed only for a specific P2P architecture, the other [10, 12, 3] assume a linearly-sorted and range-partitioned data with the option of shifting part of the data from a node to its *neighbor* and with the option of splitting a node into two half-loaded nodes.

Another way to balance the access-load is purely by replication of the high-loaded logical nodes to less loaded peers. This concept is adopted, e.g., by the HotRoD system [18] – a DHT-based architecture for interval queries.

None of the techniques currently available focus on balancing for query-paradigms that require a time-consuming and varying local processing. The strategies also put restrictions on the data domain and the system architecture.

## 3 The LOBS Balancing Framework

In this section, we propose LOBS – a general framework for load balancing of P2P structured networks with a time-consuming query-paradigm. Namely, the section describes fundamentals of LOBS and atomic operations available for balancing, the specific load measures, management of global knowledge, and the actual balancing strategies.

### 3.1 LOBS Principles and Balancing Operations

Having a closer look at the problem defined above, we can reformulate it as follows: There is a network of autonomous computational units (peers) each of which carries a certain portion of data. This network receives tasks decomposed into subtasks which are processed at individual peers depending on the specific location of the relevant data.

In this setting, the computational load of the peers can be influenced only by location of the data. In general, the data can either be *relocated* from a high-loaded peer to a peer with lesser load, or *replicated* to spread the load among several peers.

**Data Relocation** In a general similarity P2P structure, e.g. $GHT^*$ [5] or $VPT^*$ [6], neither the domain of data nor the system topology are linear and we cannot simply shift a part of the data to a neighboring node of the network. On the other hand, all the structures we focus on support a `Split` operation – creating a new node to carry approximately one half of the node's data. This leads to the following concept: Let us separate the physical peers from the logical layer of the system and allow the peers to carry more than one logical node. The

load is naturally measured per peer. Then, we can split a node that caused an overload and place the new node to a less-loaded peer. Also, this opens up the possibility of using *any* peer with a light load to relieve an overloaded peer.

More formally, the general P2P structure to be managed by LOBS is formed by a set of *nodes* $N = \{n_1, \ldots, n_k\}$. Each node consists of the local *storage* (data with a search mechanism) and the *routing structure* (links to other nodes plus the navigation strategy):

$$\forall n \in N : n = \langle data_n; R_n \rangle, \quad R_n \subseteq N \text{ is a set of links to other nodes.}$$

There is a fixed set of physical peers $P = \{p_1, \ldots, p_m\}$ and each logical node is hosted by one physical peer. Formally, there is a total mapping $h : N \longmapsto P$. Figure 2 depicts the schema described. The set of nodes hosted by a peer $p$ is also denoted as $nodes(p)$.

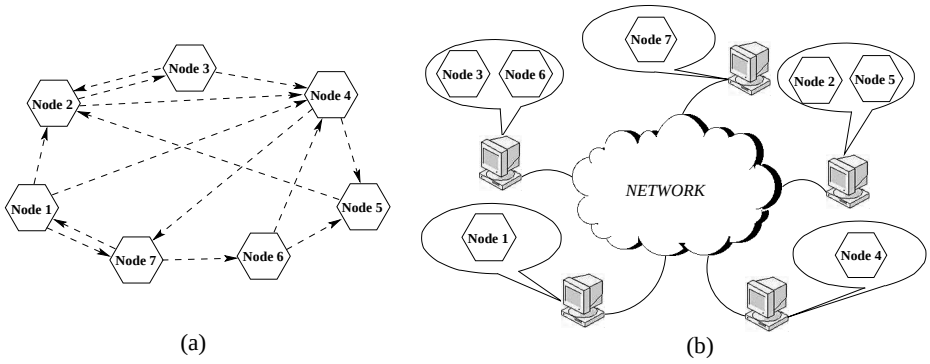


**Fig. 2.** The logical structure (a). Mapping of the nodes to the peers (b).

Thus, the following relocation operations are available for balancing: $n$.`Split`($p$) to split node $n$ creating a new node $n'$ at peer $p$, $n$.`Leave`() as a complementary operation to remove node $n$ from the structure and merge $data_n$ with an appropriate node, and operation $n$.`Migrate`($p$) to migrate node $n$ to peer $p$, $p \neq h(n)$. The `Migrate` operation requires notifying the nodes having a link to $n$ of $n$'s new location. This can be done straightforwardly in most of the structures (*GHT*$^*$ [5], *VPT*$^*$ [6], or navigation based on *CAN* [16], Skip Graphs [4]). Otherwise the internal correction mechanisms can repair the blind links in other systems (*Chord* [17], P-Grid [1]). See the technical report [15] for details.

**Data Replication** Besides regular nodes, a peer can carry one or more node replicas. Here, we have in mind a replication managed by LOBS from the outside which does not require any modifications of the particular underlying P2P structure. It is a master-slave replication where only the master node $n$ is referenced by other nodes. Any updates are first made at the master and then are spread to the slave-replicas. The search requests can be answered by any replica.

Each replica is hosted by a peer. Naturally, the node and all its replicas must be hosted by different peers. The $n.\texttt{Replicate}(p)$ operation creates a new replica of given node $n \in N$ at peer $p \in P$. Then $n.\texttt{Unify}(r)$ is a complementary operation to remove a given replica $r$ of node $n$.

### 3.2 Measuring the Load

To measure the computational load induced by the query processing, we first define $cost(\texttt{Search}(data_n))$ as the costs of processing of a single `Search` query at local data of node $n \in N$. This can be measured as the CPU time or as an approximation suitable for the specific application, e.g. the I/O costs or the number of distance computations for a costly distance measure in a metric space.

The **load** of a node is measured as the total costs of all search queries that hit the particular node in a given period of time $\Delta t$ (using principle of *sliding window*). Let $Q_n^{\Delta t}$ be the set of `Search` queries processed at node $n$ in last $\Delta t$ period of time:

$$\textbf{load}(n) = \begin{cases} \sum_{\texttt{Search} \in Q_n^{\Delta t}} cost_n(\texttt{Search}), & \text{if measured for whole } \Delta t, \\ \texttt{DontKnow}, & \text{otherwise.} \end{cases}$$

The $\textbf{load}(p)$ for a peer $p \in P$ can be defined as a sum of **load** values of its nodes:

$$\textbf{load}(p) = \sum_{n \in nodes(p)} \textbf{load}(n).$$

If any of the summands is `DontKnow` then the whole sum is `DontKnow`.

The load of a peer is influenced by: 1. processing costs of individual queries, 2. frequency of the queries hitting the peer. The first aspect can be influenced by adjusting the data volume stored at the peer. This affects the second aspect as well but a more sure way to reduce a query frequency is replication. Sometimes, it is the only way and it also requires less reorganizations.

In order to distinguish between these two causes in the case of an overload, we introduce a side load indicator **single-load** as *an average costs of a single query processing at given node or peer*. The average is taken over a certain number of last queries processed. Finally, the balancing mechanism can use information about the data volume stored by a node or the whole peer – the **data-load**. Please, see the technical report [15] for detailed definitions of the measures.

### 3.3 Global Knowledge

The P2P systems with nontrivial search paradigm typically exchange quite a high number of messages during the query processing. LOBS utilizes these messages and adds some piece of information to them in order to:

- maintain information about the average load in the system,
- exchange knowledge about the most and the least loaded peers in the system.

To calculate the global averages of the load measures, LOBS employs a slightly modified version of an algorithm [13] based on the concept of *gossips*. With a standard traffic, it maintains the averages well – even for varying load values. Information about the top (un)loaded peers is maintained by a simple mechanism [15] defined by LOBS.

Generally, we try to construct an autonomous decision-making module at every peer. Its simple work schema can be observed in Figure 3.

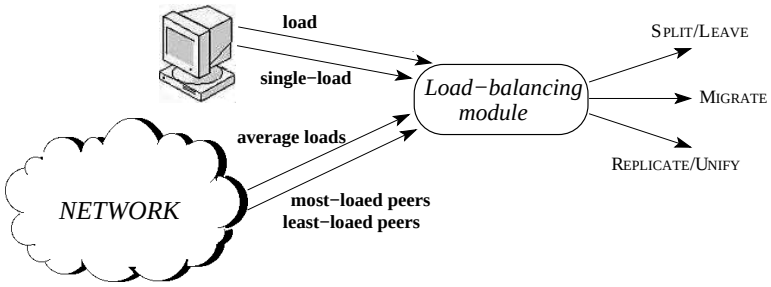


**Fig. 3.** Work schema of the load-balancing module.

### 3.4 Load Balancing

The main LOBS indicator of the load distribution quality is the *imbalance ratio* – ratio between the maximal and the minimal load of the peers. Every peer $p$ maintains current estimation of the system average load and tries to keep its load in interval $1/2 \cdot \textbf{avg-load} \leq \textbf{load}(p) \leq 2 \cdot \textbf{avg-load}$.

When $\textbf{load}(p) > 2 \cdot \textbf{avg-load}$, the balancing mechanism is activated and a suitable peer from the list of the least-loaded peers is selected to:

- either `Migrate` a node (if $|nodes(p)| \geq 2$),
- or `Split` node $n \in nodes(p)$ (if the $\textbf{single-load}(p) > 2 \cdot \textbf{avg-single-load}$),
- or `Replicate` node $\in nodes(p)$.

Vice versa, if $1/2 \cdot \textbf{avg-load} > \textbf{load}(p)$ then a peer from the list of the most-loaded peers is contacted to transfer load to $p$ using an analogous algorithm.

In addition, operation `Leave` is preferred over `Migrate` and the underloaded peer may employ the `Unify` operation, if it hosts a replica. These rules hold back the danger of a node explosion. The balancing process can affect only peers $p$ such that $\textbf{load}(p)$ differs from `DontKnow`. An occurrence of an imbalance is rechecked before an action is performed in order to overcome unimportant load fluctuations. This lead to a "conservative" and reluctant behaviour of LOBS and also to short-term imbalances but it spares costs. For full algorithms, see the report [15].

Intuitively, the balancing keeps the loads of the peers in the selected limits. The report [15] contains a proof of this fact and also a proof that the balancing process will terminate. The analysis uses the naturally-estimated influences of the balancing actions on the load of the participating peers.

## 4 Experimental Evaluation

In this section, we analyze the performance, practical impact, and costs of the load balancing using LOBS. We conducted a series of experiments in which we applied LOBS on the *M-Chord* structure [14] with a real-life data (MPEG7 features extracted from digital images compared by a special similarity function). The prototype of LOBS was implemented using the MESSIF platform [7] – a framework that supports building data structures for similarity search based on the metric space model of data. MESSIF brought us two benefits: support for the P2P approach and a straightforward linking with the *M-Chord* structure, which is implemented with MESSIF as well.

### 4.1 Experiments Setting

The basic experiments scenario is to execute a set of queries in the P2P structure and to measure various characteristics during the processing with and without the load balancing. We varied the number of peers in the network from 50 to 150 (the network of all sizes managed a fixed dataset of a million objects). We also varied the set of queries to be posed into the system. We used the basic similarity queries – the *range queries* [19] with various radii from a reasonable interval. The query objects were taken from the following three sets:

**uniform** a set of query points randomly uniformly selected from the dataset,
**zipfian** a (multi)set of queries that reflects the Zipfian distribution (power law),
**one-query** a single query repeatedly posed into the system.

While the **uniform** query set is the most natural, the **zipfian** distribution better reflects the realistic traffic. The **one-query** set induces an artificial scenario to test LOBS under extreme conditions. The size of all the query sets is 1000.

Since the experiments are performed on a structure that uses the metric space model of data, we measured the costs of the query processing as *the number of evaluations of the distance function*. This is a common indicator of the metric structure's efficiency since the CPU costs of other operations are usually practically negligible compared to the distance evaluation time.

The objectives of the experiments were 1. to observe distribution of the **load** (imbalance ratio), 2. to assess the costs of the balancing, and 3. to evaluate influence of the balancing on the system searching performance. The following section describes the specific measurements and summarizes the results.

### 4.2 Results of the Trials

The indicator we use for measuring the load distribution quality is the *imbalance ratio*. We observe the development of the maximal and the minimal load values while the system processes a set of queries. The experiment starts in a network with a fair *data distribution*, with one logical node per peer, and with no replicas. We first pose the query set to the system with the balancing off and then turn

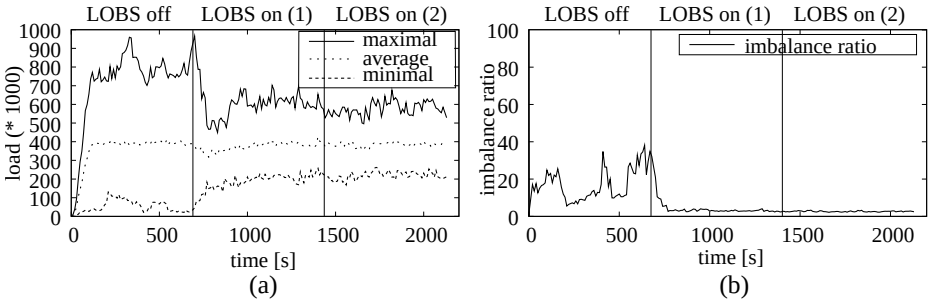


**Fig. 4.** Load development (a), imbalance ratio (b). Query set: **uniform**, 50 peers.

the balancing on and execute the same query set twice again (explained below). The results for 50 peers with the **uniform** query set are in Figure 4.

With the balancing off, the minimal and the maximal values of load differ significantly making the imbalance ratio fluctuate between 8 and 40. The LOBS makes the imbalance ratio fall down and stay very close to value 4 which is a desired behaviour. The observable fluctuations of the maximal load are caused by non-homogeneous character of the randomly-generated query set. The scenario when the balancing is turned abruptly on is not very natural and therefore we pose the same set of queries to the system once more – denoted as "LOBS on (2)". The main difference between results of LOBS on (1) and (2) are the costs of the balancing – see Table 1.

| | LOBS on (1) | LOBS on (2) |
|---|---|---|
| **Number of actions** | 13 | 2 |
| **Transferred data** | 262391 (26 %) | 43674 (4 %) |
| **Transf. data per query** | 262 | 43 |

**Table 1.** Load-balancing costs. Query set: **uniform**, 50 peers.

The table summarizes the number of balancing actions performed during query processing and the number of data items transferred by the balancing. The initial correction of the load imbalance has naturally higher requirements than a balancing process which has already been running for some time. The very low balancing costs of "LOBS on (2)" prove the property of LOBS to overcome the short-term fluctuations of the load without premature balancing. The last line of the table illustrates the average number of data objects transferred by LOBS per a processed query. Comparing this value with the average query recall (which was about 500 objects), the network traffic is increased by factor 1/2 for scenario LOBS on (1) and by factor 1/10 in the other scenario.

Figure 5 depicts the same experiment with the Zipfian distribution of the query set. We can see that the imbalance is higher than for the **uniform** query set (all imbalance ratio graphs have a ceiling of 100) but LOBS copes with the imbalance swiftly and keeps the loads stable after several initial fluctuations.

Having a higher imbalance to cope with, the costs of the balancing are slightly higher in the first run with LOBS on – 20 operations transferring 36 % of the database were performed. This increases the network traffic caused by the queries by factor 2/3. On the other hand, the costs of the LOBS on (2) scenario were even even lower than for the **uniform** query set – transferring 3 % of the data. It is caused by a higher stability of the load generated by the **zipfian** set.

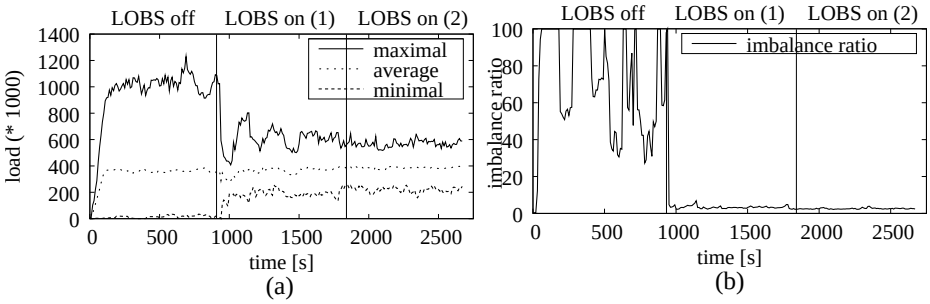


**Fig. 5.** Load development (a), imbalance ratio (b). Query set: **zipfian**, 50 peers.

The third query set used for testing represents a single query repeatedly posed into the system. Since this query hits a subset of the peers in the system, the imbalance ratio is maximal when the balancing is off – see Figure 6. LOBS keeps the imbalance ratio very stable and on the desired level. The balancing in the phase LOBS on (1) transfers 43 % of the database to reduce the imbalance. The consecutive LOBS on (2) required no balancing at all – the load induced by this query set is maximally stable.

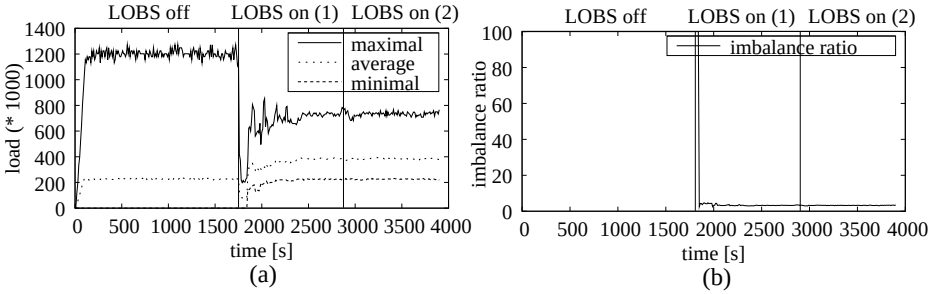


**Fig. 6.** Load development (a), imbalance ratio (b). **One-query** set, 50 peers.

All the results presented so far are for a 50-peers network. Increasing the number of peers, the trends of the load development and imbalance ratio are very similar and we do not present these graphs. The balancing costs for the **zipfian** and **one-query** sets slightly decrease as the network grows. It is caused by a finer partitioning of the dataset and better-targeted balancing actions. Costs for the **uniform** query set slightly grow with the network. In larger network, a smaller

percentage of nodes answer a single query and, therefore, a heterogeneous query traffic causes larger load fluctuations which results in higher balancing costs. See the report [15] for specific values.

Now, let us observe the influence of LOBS to utilization of the resources. Figures 7 and 8 depict the *histogram of work* measured as the number of distance computations performed by individual peers during processing of given query set. We can see that LOBS disposed of the waste of resources observable in systems without any balancing of computational load.

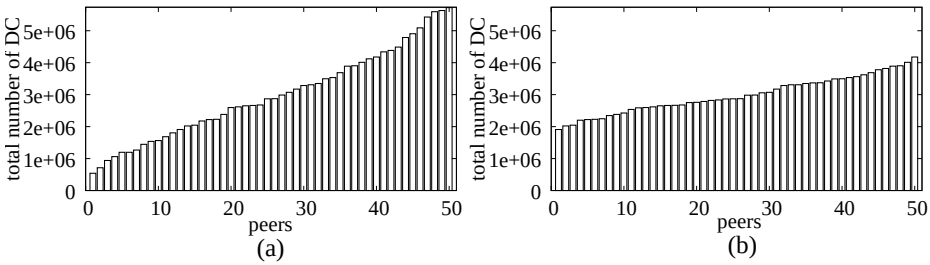


**Fig. 7.** Work histogram without LOBS (a) and with LOBS (b). Query set: **uniform**.

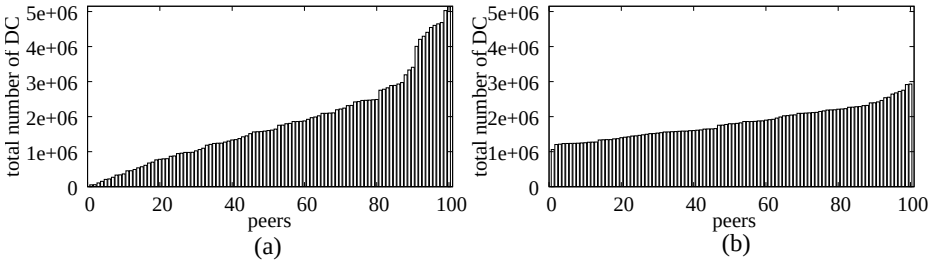


**Fig. 8.** Work histogram without LOBS (a) and with LOBS (b). Query set: **zipfian**.

Finally, we present graphs which give evidence of the query throughput of the system. Since in our testing environment the peers share the resources (mainly CPUs), the improvement of the very processing times is not significant. Therefore, we introduce a different measure to express the total processing time of a set of queries: the *Overall parallel distance computations*, which is calculated as the maximal number of metric-distance computations performed at a single peer during simultaneous processing of a the set of queries. Figure 9 depicts this value for all the query sets. We can see that values with LOBS off are significantly higher for the **zipfian** query set and the **one-query** set. Turning the LOBS on reduces the values especially significantly for these two sets. Naturally, as the number of peers grows, a set of queries is processed in a shorter time period but observe that LOBS still reduces the value by one half.

We also measured the influence on the response time of a single isolated query. For all the query sets but **one-query**, the influence was insignificant and we do not present the graphs. See the report [15] for the full experiments results.

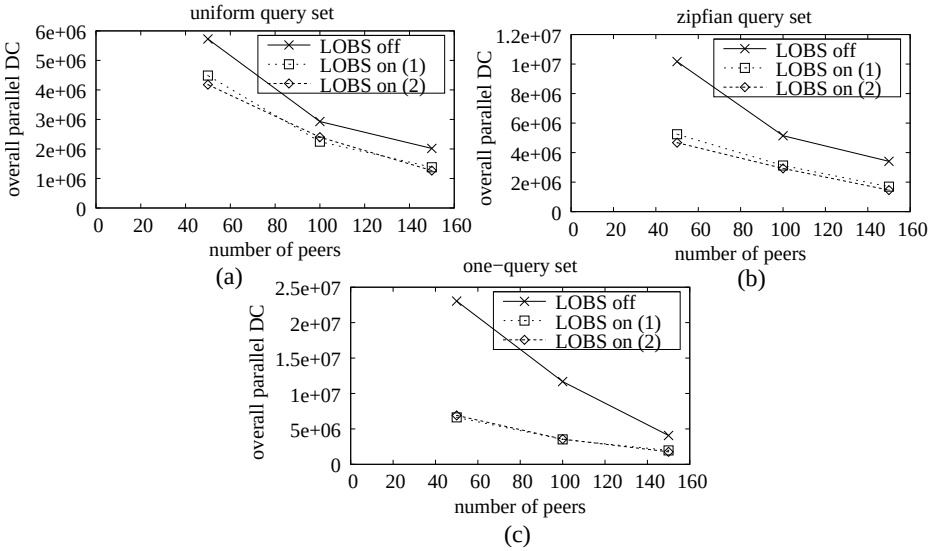


**Fig. 9.** Overall parallel costs. Query sets: **uniform** (a), **zipfian** (b), **one-query** (c).

## 5 Conclusions and Future Work

In this paper, we focused on peer-to-peer structures for efficient content-based retrieval in huge data collections. Since these systems have high computational demands, it is worth considering the computational load of individual peers. The distribution of this load is typically highly skewed, which calls for a load balancing. Existing P2P balancing techniques cannot be used and therefore we propose LOBS – a general system for load-balancing in P2P structures with the following features:

- it measures the computational load of the peers,
- it does not require a sortable data domain and linear network architecture,
- it separates the logical and physical layers of the system,
- it analyzes the load source precisely to use data relocation or replication,
- the balancing strategies are rather "conservative" and reluctant to ignore temporary load fluctuations and to avoid inefficient balancing actions.

We have implemented a prototype of LOBS and have executed a number of experiments with system *M-Chord* [14] and a real-life dataset. The results prove that LOBS is able to cope with any query-distribution and that it improves both the utilization of resources and the system performance of query processing. The costs, in terms of data transferred due to the balancing, seem to be very small in a "living system" where the balancing had time to adapt to a query-traffic. If the balancing is turned on abruptly in a non-balanced and heavily queried system, the balancing actions may transfer 25 % to 40 % of the database. These costs correspond to the level of imbalance observed in the network without balancing.

We believe that this work is a step towards applications that would efficiently manage and search up to hundreds of millions multimedia objects with a real-life query traffic. Our research group plans to build a prototype of such a distributed structure and load balancing will be an important component of the system architecture. In the future, we would like to elaborate an advanced cost model of the balancing actions. This model should be taken into consideration by the decision mechanism within each autonomous load-balancing module.

## References


1. Karl Aberer. P-Grid: A self-organizing access structure for P2P information systems. *Lecture Notes in Computer Science*, 2172:179–194, 2001.
2. Karl Aberer, Anwitaman Datta, and Manfred Hauswirth. The Quest for Balancing Peer Load in Structured Peer-to-Peer Systems. Technical report, EPFL, Swiss, 2003.
3. James Aspnes, Jonathan Kirsch, and Arvind Krishnamurthy. Load balancing and locality in range-queriable data structures. In *PODC '04: Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, pages 115–124, New York, NY, USA, 2004. ACM Press.
4. James Aspnes and Gauri Shah. Skip graphs. In *Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 384–393, 2003.
5. Michal Batko, Claudio Gennaro, and Pavel Zezula. Similarity grid for searching in metric spaces. *DELOS Workshop: Digital Library Architectures, Lecture Notes in Computer Science*, 3664/2005:25–44, 2005.
6. Michal Batko, David Novak, Fabrizio Falchi, and Pavel Zezula. On scalability of the similarity search in the world of peers. In *Proceedings of INFOSCALE 2006, Hong Kong, 2006*, pages 1–12, New York, NY, USA, 2006. ACM Press.
7. Michal Batko, David Novak, and Pavel Zezula. Messif: Metric similarity search implementation framework. In C. Thanos and F. Borri, editors, *DELOS Conference 2007: Working Notes, Pisa, 13-14 February 2007*, pages 11–23. Information Society Technologies, 2007.
8. Adina Crainiceanu, Prakash Linga, Ashwin Machanavajjhala, Johannes Gehrke, and Jayavel Shanmugasundaram. P-Ring: An index structure for peer-to-peer systems. Technical Report TR2004-1946, Cornell University, NY, 2004.
9. Fabrizio Falchi, Claudio Gennaro, and Pavel Zezula. A content-addressable network for similarity search in metric spaces. In *Proceedings of DBISP2P 2005, Trondheim, Norway, August 28–29, 2005*, pages 126–137, August 2005.
10. Prasanna Ganesan, Mayank Bawa, and Hector Garcia-Molina. Online balancing of range-partitioned data with applications to peer-to-peer systems. Technical report, Stanford U., 2004.
11. Brighten Godfrey, Karthik Lakshminarayanan, Sonesh Surana, Richard Karp, and Ion Stoica. Load balancing in dynamic structured P2P systems. In *Proceedings of INFOCOM 2004*, volume 4, pages 2253–2262, 2004.
12. David R. Karger and Matthias Ruhl. Simple efficient load balancing algorithms for peer-to-peer systems. In *SPAA '04: Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures*, pages 36–43, New York, NY, USA, 2004. ACM Press.

13. David Kempe, Alin Dobra, and Johannes Gehrke. Gossip-based computation of aggregate information. In *Proceedigs of FOCS '03*, page 482, Washington, DC, USA, 2003. IEEE Computer Society.
14. David Novak and Pavel Zezula. M-Chord: A scalable distributed similarity search structure. In *Proceedings of INFOSCALE 2006, Hong Kong, 2006*, pages 1–10, New York, NY, USA, 2006. ACM Press.
15. David Novak and Pavel Zezula. LOBS: Load balancing for similarity peer-to-peer structures. Technical Report FIMU-RS-2007-04, Faculty of Informatics, Masaryk University Brno, http://www.fi.muni.cz/reports/files/2007/FIMU-RS-2007-04.pdf, June 2007.
16. Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. In *Proceedings of SIGCOMM 2001, San Diego, California, 2001*, pages 161–172. ACM Press, 2001.
17. Ion Stoica, Robert Morris, David R. Karger, Frans M. Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of SIGCOMM 2001, San Diego, California, 2001*, pages 149–160. ACM Press, 2001.
18. Peter Triantafillou, Theoni Pitoura, and Nikos Ntarmos. Replication, load balancing and efficient range query processing in DHTs. In *Proceedings of EDBT, Munich, Germany*, 2006.
19. Pavel Zezula, Giuseppe Amato, Vlastislav Dohnal, and Michal Batko. *Similarity Search: The Metric Space Approach*, volume 32 of *Advances in Database Systems*. Springer-Verlag, 2006.